VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Adam Ludes

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VÝVOJ KALKULÁTORU PRO HODNOCENÍ ZRANITELNOSTÍ
IMPLEMENTATION OF A VULNERABILITY ASSESSMENT CALCULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Adam Ludes**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Zdeněk Martinásek, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Adam Ludes ***ID:*** 211802
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Vývoj kalkulátoru pro hodnocení zranitelností

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem bakalářské práce je návrh a implementace webové aplikace pro výpočet závažnosti zranitelnosti podle specifické metodologie v programovacím jazyce JavaScript. Grafické uživatelské rozhraní bude umožňovat zadání parametrů potřebných k výpočtu výsledného skóre a výsledky bude možné uložit do databáze. V teoretické části student provede analýzu dnes používané metodologie k hodnocení zranitelností a pozornost věnuje zejména navržené metodě v diplomové práci [1]. Dále student analyzuje možnosti Framework Vue JS. V praktické části student navrhne grafickou vizualizaci vlastního nástroje a následně implementuje za pomoci frameworku Vue JS funkční aplikaci kalkulátoru pro výpočet závažnosti zranitelností dle práce [1]. Aplikace bude také umožňovat získání výsledků uložených v databázi.

DOPORUČENÁ LITERATURA:

[1] PECL, David. Návrh metody pro hodnocení bezpečnostních zranitelností systémů. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125988. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[2] HANCHETT, Erik a Benjamin LISTWON. Vue.js in Action. Manning Publications, 2019, 375 s. ISBN 9781617294624.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se věnuje nově představenému způsobu hodnocení zranitelností, jeho srovnání s nejčastěji používanou metodou *Common Vulnerability Scoring System* (CVSS), analýze frameworku Vue.js a ostatních technologií použitých při implementaci a samotné implementaci nástroje sloužícímu k představení možností nově navržené metody.

KLÍČOVÁ SLOVA

zranitelnost, CVSS, DPSS, CVE, Vue.js, JavaScript, Python, Vue.js, Django REST Framework

ABSTRACT

This bachelor thesis focuses on a newly introduced vulnerability scoring system, compares it to a most widespread alternative, which is *Common Vulnerability Scoring System* (CVSS), analyzes the Vue.js framework and other technologies used in the implementation. Lastly it introduces an implementation of said new scoring system in a way to best showcase its capabilities.

KEYWORDS

vulnerability, CVSS, DPSS, CVE, Vue.js, JavaScript, Python, Vue.js, Django REST Framework

LUDES, Adam. *Vývoj kalkulátoru pro hodnocení zranitelností*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 62 s. Bakalářská práce. Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Adam Ludes
VUT ID autora:	211802
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Vývoj kalkulátoru pro hodnocení zranitelností

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Zdeňku Martináskovi, Ph.D. za jeho trpělivost a odborné vedení této práce. Jakubovi Janekovi chci poděkovat za jeho cenné rady při práci s Vue.js a JavaScriptem.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Bezpečnost je v informatice stále častěji zmiňovaná a s narůstajícím množstvím uživatelů, zařízení a služeb on-line roste i množství bezpečnostních zranitelností a jejich dopady. Tyto zranitelnosti mohou útočníkovi poskytnout přístup k citlivým údajům a informacím, ať už jde o jejich odcizení nebo poškození. S pomocí zranitelností je také možné zaútočit na infrastrukturu, která je nutná pro správné fungování služby.

Pro zvýšení bezpečnosti se tedy používají další nástroje jako například kontrola přístupu, šifrování, firewall a antivirové programy. Tyto faktory se podílejí na celkové bezpečnosti systému, tudíž i na celkovém dopadu, který jednotlivé zranitelnosti mají na konkrétní systémy.

V současnosti se pro hodnocení závažnosti zranitelnosti používá několik metod a jednou z nejpoužívanějších je *Common Vulnerability Scoring System* (CVSS). Metodika CVSS prošla několika verzemi, ale stále zůstává zásadní problém – příliš mnoho zranitelností je klasifikováno jako kritická, nebo vysoká, což má za následek nepřehlednost při snaze zranitelnosti odstraňovat. Z důvodu komplexnosti metody je obtížné hodnotit zranitelnosti automatizovaně a případné automatické nástroje mohou být nepřesné.

Cílem této práce je návrh a implementace webové aplikace pro výpočet závažnosti zranitelnosti za použití nově navržené metody [1], která zohledňuje použitá protiopatření za účelem vyšší přesnosti závažnosti. Metoda rozděluje faktory do logických částí které lze znovu kombinovat a tím pádem má snížené nároky pro správnnou klasifikaci a je možné ji z části automatizovat. Je zároveň mnohem jednodušší na pochopení. Implementace bude provedena v programovacím jazyce JavaScript za použití frameworku Vue.js.

V teoretické části jsou popsány metody CVSS a metoda navržená Davidem Peclem, jejich parametry a způsoby výpočtu závažnosti. Dále jsou popsány technologie použité při implementaci.

Praktická část se věnuje návrhu aplikace, představení vývojového prostředí a kompletnímu popisu fungování navržené aplikace, který je rozdělený na 2 části – Frontend a Backend. Dále bude představené grafické rozhraní aplikace včetně vzorových dat.

1 Metody hodnocení zranitelností

Zranitelnost systému je jakákoliv slabina, která při případném zneužití útočníkem může znamenat narušení bezpečnosti nebo stability zranitelného systému. Pro zjednodušení práce bezpečnostních techniků a systémových administrátorů existuje množství databází zranitelností, kde jsou jsou tyto softwarové slabiny zdokumentovány a ohodnoceny.

Ohodnocením zranitelnosti je myšleno vypočítání skóre, které popisuje její závažnost, tzn. dopad této zranitelnosti na systém v případě jejího zneužití. Administrátoři a bezpečnostní technici potom mohou prioritizovat závažnější zranitelnosti nad těmi méně závažnými, což zkvalitňuje jejich práci.

Pro ohodnocení zranitelnosti bylo vyvinuto několik metod, z těch nejpoužívanějších si představíme metodu *Common Vulnerability Scoring System* (CVSS)[2] a metodu navrženou Davidem Peclem [1].

1.1 Common Vulnerability Scoring System

CVSS je otevřený standard pro výpočet závažnosti zranitelnosti, vytvořený neziskovou organizací MITRE a spravovaný neziskovou organizací *Forum of Incident Response and Security Teams* (FIRST). Díky jeho otevřenosti a jednoduchosti je nyní hojně využívaný při hodnocení nově objevených zranitelností.

Skóre zranitelnosti se ukládá do databází, ze kterých pak čerpají vývojáři zranitelného softwaru a výrobci monitorovacích nástrojů, tyto nástroje používají bezpečnostní technici a administrátoři k odhalení zranitelností na svých zařízeních a aplikacích. Díky nim se mohou soustředit více na jejich odstraňování, místo jejich hledání. Aktuálně nejnovější CVSS je verze 3.1.

1.1.1 Princip CVSS

Metoda používá tři oblasti hodnocení: Base (Základní skóre), Temporal (Aktuální skóre) a Environmental (Skóre prostředí). Každá oblast se zabývá jinou skupinou faktorů, které mají vliv na konečný dopad zranitelnosti. [3]

Základní skóre se zabývá pouze vlastní charakteristikou zranitelnosti. Ta se nemění s časem ani s prostředím, ve kterém se zranitelnost může vyskytnout. Jeho faktory jsou rozdělené na 2 části: metriky exploitace a metriky dopadu. Metriky exploitace odráží technické prerekvizity zneužití zranitelnosti, tzn. jak jednoduše je možné zranitelnost zneužít. Metriky dopadu odráží, čeho je možné při zneužití zranitelnosti dosáhnout.

Aktuální skóre definuje charakteristiku zranitelnosti, která se s časem může měnit, například je-li slabina opravena v aktualizaci, nebo byl-li objeven jednoduše použitelný nástroj k jejímu zneužití.

Skóre prostředí reprezentuje vlastnosti zranitelnosti, které přísluší určitému systému, na kterém se zranitelnost nachází. Definuje požadavky na triádu *Confidentiality, Integrity, Availability* (CIA) – důvěrnost, integrita, dostupnost a modifikované faktory exploitace, které se mohou od základního skóre lišit např. přítomností bezpečnostních systémů, které brání přístupu ze zařízení mimo lokální sítě.

1.1.2 Výpočet skóre

Ze všech tří oblastí hodnocení je povinná pouze ta základní. Aktuální skóre a skóre prostředí není nutné vyplnit, ale výrazně přispívají na přesnosti pro konkrétní systém v konkrétním čase. Aktuální skóre je výsledek základního skóre obohacený o časově závislé parametry. Skóre prostředí potom vychází z parametrů, které platí na konkrétním stroji a časově závislých parametrů.

Základní skóre

Základní skóre je hodnota uváděná u nově objevených zranitelností. Z tohoto skóre vychází hodnocení závislá na době a na konkrétním stroji. Pro výpočet skóre je nutné znát parametry a jejich možné hodnoty, viz tabulka 1.1.

Rozsahem je myšlen rozsah pravomocí před a po využití exploitu. Např. pokud zranitelnost dovolí útočníkovi uniknout z virtualizovaného prostředí a ovlivnit i hostitelský systém, rozsah by byl změněný.

Tab. 1.1: Základní parametry CVSS [3]

Parametr skóre	Možnosti	Číselné hodnoty
Vektor útoku	Vnější síťový	0,85
	Vnitřní síťový	0,62
	Lokální	0,55
	Fyzický	0,20
Komplexita útoku	Nízká	0,77
	Vysoká	0,44
Vyžadovaná oprávnění	Žádná	0,85
	Nízká	0,62[1]
	Vysoká	0,27[2]
Interakce uživatele	Žádná	0,85
	Vyžadována	0,62
Rozsah	Změněný	
	Nezměněný	
Dopad na důvěrnost/integritu/dostupnost	Vysoký	0,56
	Nízký	0,22
	Žádný	0,00

Základní skóre se počítá pomocí několika podvzorců. Nejprve vypočítáme skóre dopadu:

$$ZakladDopadu = 1 - [(1 - Duvernost) \cdot (1 - Integrita) \cdot (1 - Dostupnost)] \quad (1.1)$$

Pokud je rozsah nezměněný, použije se následující rovnice:

$$Dopad = 6,42 \cdot ZakladDopadu \quad (1.2)$$

Pokud je rozsah změněný, rovnice má následující tvar:

$$Dopad = 7,52 \cdot (ZakladDopadu - 0,029) - 3,25 \cdot (ZakladDopadu - 0,02)^{15} \quad (1.3)$$

Dále je nutné vypočítat skóre exploitability:

$$\begin{aligned} Exploitabilita = 8,22 \cdot VektorUtoku \cdot KomplexitaUtoku \\ \cdot VyzadovanaOpravnení \cdot InterakceUzivatele \end{aligned} \quad (1.4)$$

[1] 0.68 pokud je Rozsah změněný

[2] 0.5 pokud je Rozsah změněný

Konečný výpočet skóre je znovu závislý na hodnotě parametru *Rozsah*. Pro nezměněný platí rovnice:

$$ZakladniSkore = Roundup(Minimum[(Dopad + Exploitabilita); 10]) \tag{1.5}$$

Roundup – Zaokrouhlení čísla vždy nahoru s přesností na 1 desetinné místo
Minimum – Vrátí menší ze dvou hodnot

Pro změněný *Rozsah* platí rovnice:

$$\begin{aligned} ZakladniSkore = Roundup(Minimum[\\ 1,08 \cdot (Dopad + Exploitabilita); 10]) \end{aligned} \tag{1.6}$$

Aktuální skóre

Aktuální skóre měří aktuální stav zranitelnosti, jako je existence exploitu, dostupnost informací o zranitelnosti a existenci bezpečnostních aktualizací.

K výpočtu aktuálního skóre potřebujeme znát parametry a jejich možné hodnoty, viz tabulka 1.2.

Tab. 1.2: Aktuální parametry CVSS [3]

Parametr aktuálního skóre	Možnosti	Číselné hodnoty
Kvalita exploitu	Vysoká kvalita	1,00
	Funkční	0,97
	Proof-of-Concept	0,94
	Bez důkazů	0,91
	Není definováno	1,00
Úroveň nápravy	Nedostupná záplata	1,00
	Workaround	0,97
	Dočasná záplata	0,96
	Oficiální záplata	0,95
	Není definováno	1,00
Dostupnost informací	Veřejná	1,00
	Přiměřená	0,96
	Nízká	0,92
	Není definováno	1,00

Rovnice pro výpočet aktuálního skóre má následující tvar:

$$\begin{aligned} AktualniSkore = Roundup(ZakladniSkore \cdot KvalitaExploitu \\ \cdot UrovenNapravy \cdot DostupnostInformaci) \end{aligned} \tag{1.7}$$

Skóre prostředí

Skóre prostředí umožňuje přizpůsobení skóre konkrétnímu systému podle požadavků, které jsou na systém kladeny, a podpůrných systémů, které ho pomáhají chránit. Kromě parametrů požadavků na CIA triádu obsahuje modifikované parametry základního skóre, protože jsou to parametry daného prostředí a díky různým nastavením zabezpečení mohou mít hodnoty jiné než základní skóre. Skóre tak přizpůsobuje vlastnostem lokálního prostředí.

Kromě těchto „modifikovaných" základních parametrů je nutná ještě trojice parametrů požadavků na důvěrnost, integritu a dostupnost. Mají stejné možnosti a hodnoty, proto jsou v tabulce 1.3 sloučeny.

Tab. 1.3: Parametry prostředí CVSS [3]

Parametr prostředí skóre	Možnosti	Číselné hodnoty
Požadavky na důvěrnost/integritu/dostupnost	Vysoké	1,50
	Střední	1,00
	Nízké	0,50
	Není definováno	1,00
Modifikované základní parametry	totožné s tab. 1.1	

K výpočtu skóre prostředí použijeme Modifikovaný dopad:

$$\begin{aligned} ModifZakladDopadu = Minimum(1 \\ &- [(1 - PozadavekDuvernosti \\ &\cdot ModifDuvernost) \\ &\cdot (1 - PozadavekIntegrity \\ &\cdot ModifIntegrita) \\ &\cdot (1 - PozadavekDostupnosti \\ &\cdot ModifikovanaDostupnost)]; 0,915) \end{aligned} \tag{1.8}$$

Pokud je modifikovaný rozsah nezměněný, použije se následující rovnice:

$$ModifDopad = 6,42 \cdot ModifZakladDopadu \tag{1.9}$$

Pokud je rozsah změněný, rovnice má následující tvar:

$$\begin{aligned} ModifDopad = 7,52 \cdot (ModifZakladDopadu - 0,029) \\ - 3,25 \cdot (ModifZakladDopadu \cdot 0,9731 - 0,02)^{13} \end{aligned} \tag{1.10}$$

Vypočítáme modifikovanou exploitabilitu:

$$\begin{aligned} ModifExploitabilita = 8,22 &\cdot ModifVektorUtoku \\ &\cdot ModifKomplexitaUtoku \\ &\cdot ModifVyzadovanaOprávnění \\ &\cdot ModifInterakceUzivatele \end{aligned} \tag{1.11}$$

Konečný výpočet skóre je znovu závislý na hodnotě parametru Rozsah. Pro nezměněný platí rovnice:

$$\begin{aligned} SkoreProstredi = Roundup(Roundup[Minimum(& \\ & [ModifDopad + ModifExploitabilita]; 10)] \\ & \cdot KvalitaExploitu \cdot UrovenNapravy \\ & \cdot DostupnostInformaci) \end{aligned} \tag{1.12}$$

Pro změněný rozsah platí rovnice:

$$\begin{aligned} SkoreProstredi = Roundup(Roundup[Minimum(1,08 & \\ & \cdot [ModifDopad + ModifExploitabilita]; 10)] \\ & \cdot KvalitaExploitu \cdot UrovenNapravy \\ & \cdot DostupnostInformaci) \end{aligned} \tag{1.13}$$

1.1.3 Výhody a nevýhody metody CVSSv3

Jednou z velkých výhod této metody je její komplexnost. Znamená to, že výsledek přesně ohodnotí dopad na konkrétní systém, to platí ale pouze v případě, kdy se vypočítá skóre prostředí. Navíc tato komplexní metoda pro správné zvolení parametrů potřebuje obsáhlé znalosti vlastností konkrétního systému, proto většinou není možné hodnotit skóre prostředí s pomocí automatizovaných nástrojů. Alespoň ne pro získání přesných hodnot, které vypovídají o dopadu dané zranitelnosti na konkrétní systém.

Mimo to ještě mohou být některé konkrétní případy zranitelností hodnoceny nepřesně, kvůli způsobu výpočtu. Například v případě, že zranitelnost ovlivní pouze dostupnost, zatímco důvěrnost a integrita zůstanou zachovány. CVSS skóre v tomto případě nemusí odrážet dopad zranitelnosti přesně. Základní skóre by bylo poměrně nízké, ale zranitelnost stále může mít fatální následky například v případě distribuce elektrické energie, kde nejvyšší prioritu má právě dostupnost. Je klíčové, aby nebyly přerušeny dodávky elektřiny, zatímco integrita dat a důvěrnost hrají menší roli.

Omezené využití automatizace při prioritizaci zranitelností tak ústí v monitorovací nástroje, které často používají pouze základní, nebo aktuální skóre, ve kterém se ale neodráží vlastnosti konkrétního stroje, to je ponecháno na administrátorovi.

Případně mohou využívat i skóre prostředí, ale takové nástroje budou náročné na údržbu dat a budou potřebovat více zásahů od techniků a administrátorů. Některé nástroje mohou ještě používat své vlastní způsoby hodnocení v kombinaci s CVSS ve snaze odstranit její nevýhody.

Její další velkou výhodou je její historie. CVSSv3 je třetí verze a každá další verze přináší úpravy, které vedou k celkovému zlepšení této metody a přesnějším výsledkům. Bohužel je ale stupnice CVSS pouze od 1 do 10 s jedním desetinným místem a proto nemá stupnice velké rozlišení na porovnání rozdílů.

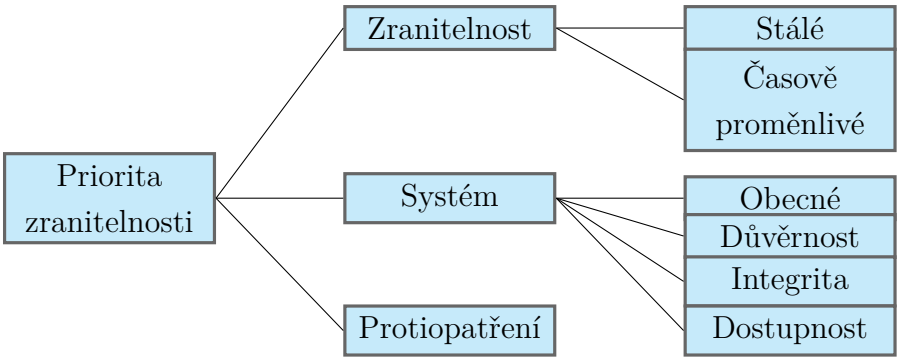
1.2 Metoda navržená Davidem Peclem

Metodu, která je v implementaci použita, navrhl ve své diplomové práci David Pecl [1]. Cílem práce byl návrh metodologie, která je přehledná, ale zároveň má kvalitní výstup, tzn. kvalita prioritizace podle této metody nemá být zanedbána. Metoda Davida Pecla – *David Pecl Scoring System* (DPSS), vychází z CVSS, pro lepší rozlišení používá stupnici 0-1000 a parametry, které jsou jednodušší na pochopení. Zakládá se na CIA triádě, přičemž požadavky a dopad na prvky CIA se promítají do výsledného hodnocení.

1.2.1 Princip metody

Metoda, podobně jako skóre prostředí u CVSS, nepoužívá k výpočtu pouze informace o zranitelnosti jako takové, ale zaměřuje se i na faktory proměnlivé s časem a protiopatření použité na systému. Díky tomu je výsledek přizpůsobený aktuálním podmínkám na konkrétním stroji a prioritizace množství zranitelností na různých strojích je přesnější. Zároveň jsou všechny parametry pro člověka čitelné a srozumitelné.

K výpočtu skóre se používají parametry ze tří okruhů – zranitelnost, systém a protiopatření, viz obrázek 1.1. [1] Všechny 3 skupiny parametrů jsou k výpočtu nutné, na rozdíl od CVSS. Zabraňuje to nepřesnosti vznikající v případě, že nejsou započítané vlastnosti systému a použitá protiopatření. Zároveň má skóre vyšší přesnost v případech, kdy má zranitelnost vliv pouze na jeden z prvků CIA, protože dopad je započítán jednotlivě.



Obr. 1.1: Schéma skupin parametrů DPSS [1]

1.2.2 Parametry zranitelnosti

Hodnocení zranitelnosti popisuje pouze zranitelnost jako takovou, podobně jako základní skóre u CVSS. Na rozdíl od CVSS ale dělí svoje parametry podle časové závislosti na stálé a časově proměnlivé.

Stálé parametry

Tabulka 1.4 představuje parametry dopadu zranitelnosti na CIA triádu. To vyjadřuje, jak velkou hrozbu zranitelnost představuje pro zranitelný systém a data na něm přítomná.

Dopad na důvěrnost charakterizuje množství a citlivost dat, která mohou být s pomocí zranitelnosti odcizena, dopad na integritu se věnuje poškození dat, které může útočník s pomocí zranitelnosti způsobit a dopad na dostupnost popisuje možnost vyřazení služeb z provozu a jejich důležitost.

Mezi stálé parametry v tabulce 1.5 se dále řadí požadovaná oprávnění, obtížnost zneužití a potřebná interakce uživatele. Jsou to požadavky na zdatnost útočníka a další požadavky, které by mohly úspěšnému zneužití bránit.

Časově proměnlivé parametry

Tyto parametry odráží aktuální vývoj situace, jako jsou povědomí o zranitelnosti, existenci exploitu a aktuální zneužívání zranitelnosti, které zachycují monitorovací služby (Threat Intelligence). Parametry nejsou povinné, proto se v případě, že není známé, která z možností platí, použije hodnota 0,5.

Tab. 1.4: Dopad zranitelnosti na CIA triádu [1]

Parametr	Možnosti	Číselné hodnoty
Dopad na důvěrnost	Odcizení všech dat	1,00
	Rozsáhlé množství odcizených kritických dat	0,78
	Rozsáhlé množství odcizených necitlivých dat	0,67
	Minimální množství odcizených kritických dat	0,67
	Minimální množství odcizených necitlivých dat	0,22
Dopad na integritu	Úplné poškození všech dat	1,00
	Rozsáhlé množství vážně poškozených dat	0,78
	Rozsáhlé množství lehce poškozených dat	0,56
	Minimální množství vážně poškozených dat	0,33
	Minimální množství lehce poškozených dat	0,11
Dopad na dostupnost	Úplná nedostupnost všech služeb	1,00
	Rozsáhlá nedostupnost primární služby	0,78
	Rozsáhlá nedostupnost sekundární služby	0,56
	Minimální nedostupnost primární služby	0,56
	Minimální nedostupnost sekundární služby	0,11

Tab. 1.5: Další stálé parametry zranitelnosti [1]

Parametr	Možnosti	Číselné hodnoty
Obtížnost zneužití	Velmi nízká	1,00
	Nízká	0,78
	Vysoká	0,44
	Velmi vysoká	0,00
Požadovaná oprávnění	Žádná	1,00
	Nízká	0,80
	Vysoká	0,40
Interakce uživatele	Žádná	1,00
	Vyžadována	0,40

Tab. 1.6: Časově proměnlivé parametry zranitelnosti [1]

Parametr	Možnosti	Číselné hodnoty
Dostupnost informací	Veřejně známá	1,00
	Známá	0,67
	Tajná	0,44
	Neznámá	0,11
	Nelze určit	0,50
Exploitace	Pomocí automatizovaných nástrojů	1,00
	Jednoduchá	0,56
	Proof-of-Concept	0,33
	Teoretická	0,11
	Nelze určit	0,50
Threat Intelligence	Velmi vysoké	1,00
	Vysoké	0,75
	Střední	0,50
	Nízké	0,25
	Žádné	0,00
	Nelze určit	0,50

1.2.3 Parametry systému

Hodnocení systému se věnuje vlastnostem zranitelného zařízení, datům na něm přítomným, bezpečnostním požadavkům na něj kladeným a dopadu při jejich porušení.

Informace o systému

Tyto parametry vyjadřují povahu systému a dat na něm uloženým. Parametry vztahu společnosti k datům nejsou započítány do výsledného skóre, slouží pouze pro upřesnění informací o systému.

Požadavky na CIA

Těchto parametrů je velké množství a je nutné je ustanovit pro každý z prvků CIA (důvěrnost/integrita/dostupnost) zvlášť. Tabulky 1.8 a 1.9 tedy obsahují parametry, které jsou pro všechny prvky CIA stejné, ale je nutné je ustanovit pro každý prvek zvlášť.

Tab. 1.7: Obecné parametry systému [1]

Parametr	Možnosti	Číselné hodnoty
Typ dat	Citlivá osobní data	1,00
	Osobní data	0,80
	Jiná data	0,50
Data se vztahují k	Zákazníkům společnosti	1,00
	Zaměstnancům společnosti	0,90
	Jiným fyzickým nebo právnickým osobám	0,80
Systém je produkční prostředí	Ano	1,00
	Ne	0,00
Systém je dostupný z internetu	Ano	1,00
	Ne	0,00
Data patří této společnosti	Ano	Nezapočítává se
	Ne	
Firma zpracovává tato data	Ano	Nezapočítává se
	Ne	
Firma dlouhodobě ukládá tato data	Ano	Nezapočítává se
	Ne	

Tab. 1.8: Legislativní a finanční dopady narušení [1]

Parametr	Možnosti	Číselné hodnoty
Narušení porušuje legislativu	Ano	1,00
	Ne	0,00
Narušení porušuje interní předpisy	Ano	1,00
	Ne	0,00
Narušení porušuje jiná nařízení	Ano	1,00
	Ne	0,00
Existuje smluvní zavázání zajištění důvěrnosti/integrity/dostupnosti	Ano	1,00
	Ne	0,00
Finanční dopad při narušení	Bankrot	1,00
	významný vliv na roční zisk	0,75
	menší vliv na roční zisk	0,50
	minimální	0,25
	žádný	0,00
Narušení má finanční dopad na firmu vlastnící data	Ano	1,00
	Ne	0,00
Narušení má finanční dopad na jinou společnost	Ano	1,00
	Ne	0,00

Tab. 1.9: Další dopady při narušení a požadavky na CIA [1]

Parametr	Možnosti	Číselné hodnoty
Dopad na pověst při narušení	Poškození celé značky	1,00
	Poškození dobrého jména společnosti	0,75
	Ztráta klíčových zákazníků	0,50
	Minimální	0,50
	Žádný	0,50
Narušení má dopad na pověst firmy vlastnící data	Ano	1,00
	Ne	0,00
Narušení má dopad na jinou společnost	Ano	1,00
	Ne	0,00
Narušení má vliv na zaměstnance	Ano	1,00
	Ne	0,00
Procesy ve firmě ovlivněné narušením	Kritické	1,00
	Významné, týkající se jiných aktivit	0,75
	Významné, týkající se obchodních aktivit	0,50
	Nevýznamné	0,25
	Žádné	0,00
Narušení má kromě této firmy vliv na	Širokou veřejnost	1,00
	Zákazníky	0,80
	Nikoho	0,00
Počet ovlivněných subjektů	Miliony	1,00
	Statisíce	0,95
	Tisíce	0,90
	Stovky	0,70
	Desítky	0,30
	Jednotky	0,10
	Nikdo	0,00
Požadavky na důvěrnost/integritu/ dostupnost	Kritické	1,00
	Vysoké	0,75
	Střední	0,50
	Nízké	0,25
	Žádné	0,00

1.2.4 Parametry protiopatření

Hodnocení protiopatření charakterizuje zabezpečení zranitelného systému. Jedná se o dodatečné služby, které tlumí dopady zranitelností. V případě, že služba není implementována, je číselná hodnota parametru 1,00.

Tab. 1.10: Parametry protiopatření [1]

Parametr	Implementováno	Číselná hodnota
Šifrování	Ano	0,80
Řízení přístupu	Ano	0,80
Firewall	Ano	0,90
Mikrosegmentace	Ano	0,90
Antivirový systém	Ano	0,90
Nástroje pro kontrolu integrity	Ano	0,80
Správa oprávnění	Ano	0,90
Logování	Ano	0,90
Zálohování	Ano	0,90[5]
Vysoká dostupnost	Ano	0,80
DoS ochrana	Ano	0,90

1.2.5 Výpočet skóre

Pro výpočet skóre nejprve vypočítáme vážený dopad na prvky CIA triády.

Vážený dopad na důvěrnost

$$\begin{aligned} Duvernost = [&Sum(ObecneParametrySystemu) \\ &+ Sum(PozadavkyDuvernostiSystemu)] \div 19 \end{aligned} \tag{1.14}$$

Sum – Součet všech parametrů tabulky
ObecneParametrySystemu – tabulka 1.7
PozadavkyDuvernostiSystemu – tabulka 1.8, v tomto případě pro důvěrnost

Pokud je součet požadavků na důvěrnost roven nule, nejsou na důvěrnost daného systému kladeny žádné nároky a tudíž bez ohledu na součet obecných parametrů systému se *Duvernost* musí také rovnat nule.

$$Sum(PozadavkyDuvernostiSystemu) = 0 \implies Duvernost = 0 \tag{1.15}$$

[5]Při výpočtu váženého dopadu na dostupnost se použije hodnota 0,80

$$ProtiopatreniDuvernosti = Sifrovani \cdot RizeniPristupu \cdot Firewall \cdot Mikrosegmentace \cdot Antivirus \tag{1.16}$$

$$VazenyDopadNaDuvernost = Duvernost \cdot ProtiopatreniDuvernosti \cdot DopadNaDuvernost \tag{1.17}$$

DopadNaDuvernost - dopad na důvěrnost z parametrů zranitelnosti z tabulky 1.4

Vážený dopad na integritu

$$Integrita = [Sum(ObecneParametrySystemu) + Sum(PozadavkyIntegritySystemu)] \div 19 \tag{1.18}$$

Zde platí stejná podmínka jako u důvěrnosti:

$$Sum(PozadavkyIntegritySystemu) = 0 \Longrightarrow Integrita = 0 \tag{1.19}$$

$$ProtiopatreniIntegrity = NastrojeKontrolyIntegrity \cdot RizeniPristupu \cdot SpravaOpravneni \cdot Logovani \cdot Zalohovani \tag{1.20}$$

$$VazenyDopadNaIntegritu = Integrita \cdot ProtiopatreniIntegrity \cdot DopadNaIntegritu \tag{1.21}$$

Vážený dopad na dostupnost

$$Dostupnost = [Sum(ObecneParametrySystemu) + Sum(PozadavkyDostupnostiSystemu)] \div 19 \tag{1.22}$$

Opět platí podmínka vysvětlená dříve:

$$Sum(PozadavkyDostupnostiSystemu) = 0 \Longrightarrow Dostupnost = 0 \tag{1.23}$$

$$ProtiopatreniDostupnosti = VysokaDostupnost \cdot Zalohovani \cdot DoSOchrana \tag{1.24}$$

$$VazenyDopadNaDostupnost = Dostupnost \cdot ProtiopatreniDostupnosti \cdot DopadNaDostupnost \tag{1.25}$$

Vážené skóre zranitelnosti

Dále se vypočítá skóre zranitelnosti. Protože má každý parametr zranitelnosti stanovenou jinou váhu, je nutné je svojí váhou vynásobit a až potom je sečíst. V následujícím výpočtu používáme parametry z tabulek 1.5 a 1.6.

$$\begin{aligned} VazeneSkoreZranitelnosti = {} & ThreatIntelligence \\ & + (Exploitace \cdot 0,8) \\ & + (DostupnostInformaci \cdot 0,4) \\ & + (InterakceUzivatele \cdot 0,7) \\ & + (PozadovaneOpravneni \cdot 0,7) \\ & + (ObtiznostZneuziti \cdot 0,5) \end{aligned} \tag{1.26}$$

Výsledná priorita zranitelnosti

Pro výpočet konečné priority zranitelnosti si nejprve vypočítáme vážené skóre:

$$\begin{aligned} VazeneSkore = {} & VazeneSkoreZranitelnosti \\ & \cdot (VazenyDopadNaDuvernost \\ & + VazenyDopadNaIntegritu \\ & + VazenyDopadNaDostupnost) \end{aligned} \tag{1.27}$$

Konečná priorita zranitelnosti je vyjádřena jako poměr váženého skóre a maximálního možného váženého skóre (12,3). Následně je pro lepší rozlišení hodnota vynásobena tisíci a hodnota je zaokrouhlena na celé číslo.

$$PrioritaZranitelnosti \doteq \frac{VazeneSkore}{12,3} \cdot 1000 \tag{1.28}$$

1.2.6 Výhody a nevýhody DPSS

Metoda díky rozdělení parametrů do logických okruhů může být využita způsobem, kdy lze jednotlivé celky znovu použít. Například v případě, kdy je jedna zranitelnost přítomna na více systémech, nebo má jeden systém více zranitelností, jedna sada protiopatření je využita na několika systémech. To značně ulehčuje práci administrátorům a technikům, kteří nemusí pracně upravovat několik záznamů v případě jakýchkoliv změn nebo přidání nových záznamů.

Protože tento modulární systém umí být responzivní, je například možné automatizovaně přepočítat skóre pro každou zranitelnost, která je na daném systému přítomna, v případě změny vlastností tohoto systému.

V případě, že by existovaly centrální databáze s vlastnostmi objevovaných zranitelností, stačilo by pouze do firemního systému uvést, který systém má jakou zranitelnost a jejich vlastnosti by se mohly dynamicky dostávat z té centrální, což by více zjednodušilo celý proces.

Velkou nevýhodou této metody je fakt, že se jedná o úplně nový způsob hodnocení, který nebyl vyzkoušen v reálném prostředí a neexistují databáze a programy které by nasazení umožnily a usnadnily. To má za úkol změnit tato práce, která má sloužit k představení této metody ve webovém rozhraní.

2 Technologie použité v implementaci

Aplikace je rozdělená na frontend, který představuje grafické webové rozhraní aplikace, a backend, který je na straně serveru a stará se o výpočty a ukládání dat do databáze. Pro pochopení fungování aplikace je nutné představit technologie, se kterými pracujeme.

2.1 JavaScript

JavaScript (JS) je interpretovaný, objektově orientovaný programovací jazyk známý především svým využitím na webových stránkách. Využívá se ale i mimo prostředí webového prohlížeče. Například framework Electron využívá JavaScript společně s HTML a CSS k tvorbě desktopových aplikací, jako je například Visual Studio Code, WhatsApp a Slack. [4] [5]

Ve webovém prostředí se používá k návrhu a ovlivnění chování vzhledu webových stránek, například závislém na událostech, jako je kliknutí na tlačítko nebo změna dat. V této implementaci používáme JavaScript ve frameworku Vue.js právě ve webovém prostředí.

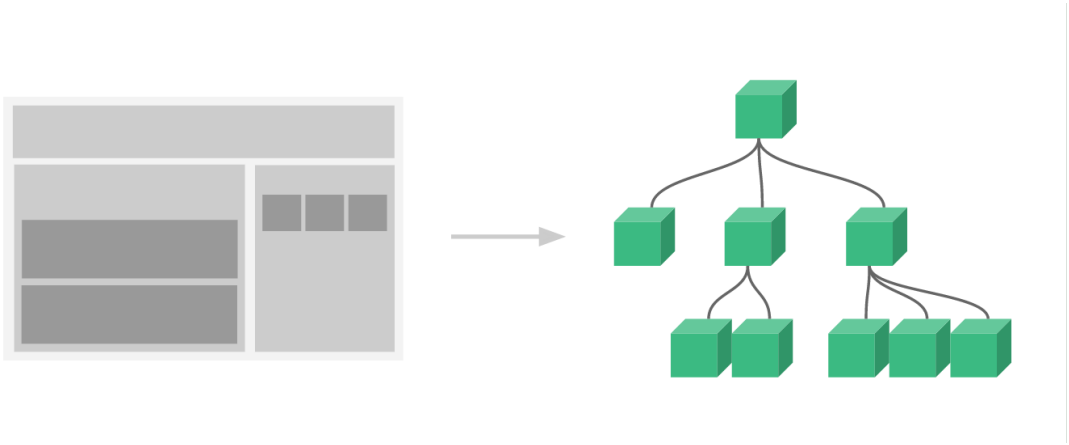
2.2 Framework Vue.js

Vue.js (často označovaný pouze jako Vue) je progresivní frontendový framework napsaný v JavaScriptu. Vytvořil jej Evan You po zkušenostech získaných díky zaměstnání v Google. Chtěl tak převzít to, co se mu líbilo na frameworku Angular, přičemž chtěl zachovat framework co nejlehčí: *„For me, Angular offered something cool which is data binding and a data driven way of dealing with a DOM, so you don't have to touch the DOM yourself.“* [6]

Základní knihovna je velmi omezená a neobsahuje funkce navíc, které by ji zvětšovaly. Samotný framework je proto velmi lehký a všechny potřebné dodatečné funkce zajišťují rozšiřující knihovny. Díky této progresivitě a modularitě je ho možné zasadit do již fungujícího řešení, nebo s pomocí jeho nástrojů postavit řešení celé.

Vue umožňuje skládat výsledný vzhled s pomocí komponentů, každý komponent má svůj vlastní soubor s příponou .vue a obsahuje tři části (zvané elementy), ty tvoří strukturu, chování a vzhled. První element se nazývá `template` a představuje HTML šablonu, do které je možné vkládat další komponenty. Další element je `script`, kam je možné psát JavaScriptový kód. Obsahuje výraz *export default {}*, ve kterém se definují data komponentu a jeho metody. Posledním elementem je `style`, ve kterém se definují CSS styly specifické pro komponent, tzn. tvoří se zde grafický vzhled. [7]

Díky možnosti vkládat Vue komponenty do jiných komponentů je možné tvořit výsledný vzhled na základě hierarchie, viz obrázek 2.1



Obr. 2.1: Vue.js hierarchie komponentů, převzato z [7]

V této implementaci pracujeme s několika rozšiřujícími knihovnami, které zajišťují správné fungování aplikace a usnadňují její návrh.

Vue Router

Vue Router [8] se používá pro Client-side routing. V případě klasického Server-side routingu každý odkaz odesílá dotaz na server a následně se přijímá celá stránka, což může znamenat zpomalení z důvodu nutnosti načíst znovu komponenty, které už načtené byly.

Client-side routing umožňuje renderování aplikace na různých definovaných podstránkách přímo na straně klienta. Změnu URL router zaznamená a stránku příslušně změní. Client-side Router také zjednodušuje navigaci mezi předešlými stavy stránky, což by u Server-side routingu znamenalo obnovení celé stránky a tudíž nové posílání dat která už načtena jsou. [9]

Díky knihovně Vue Router také můžeme definovat parametrické URL adresy, kde se zobrazená data mění v závislosti na adrese. Například v případě adresy */system/:id* se vždy zobrazí stránka systému, ale zobrazená data se budou lišit v závislosti na použitém ID.

Vuex

Vuex [10] je knihovna sloužící ke správě dat na centrální místě, místo toho aby si data spravoval každý komponent. Pro každou webovou aplikaci jsou data to nejdůležitější a Vuex práci s nimi výrazně zjednodušuje. Díky této knihovně není nutné si data

mezi jednotlivými komponentami předávat manuálně, což by značně celou aplikaci zkomplikovalo. Místo toho se data ukládají do centrálního místa, kde k nim mají všechny komponenty přístup, pokud ho potřebují. Umožňuje psát reaktivní aplikace která se responzivně mění společně s načtenými daty.

Axios

Axios [11] je HTTP klient napsaný v jazyce JavaScript. Používá se pro komunikaci frontendu s programovým rozhraním – *Application Programming Interface* (API) backendu. Jsme díky němu tedy schopní odesílat vyplněné formuláře, které backend uloží do databáze.

Element Plus

Element Plus [12] je knihovna obsahující velké množství komponentů pro uživatelská rozhraní k jednodušší implementaci webových aplikací. Tyto komponenty mají jednoduchý vzhled a vzájemně se k sobě hodí. Knihovna byla použita kvůli jejím tabulkám ve kterých lze jednoduše vyhledávat a dobře vypadajícím formulářům.

2.3 Python

Python je podobně jako JavaScript interpretovaný a objektově orientovaný programovací jazyk. Má velmi jednoduchou a čitelnou syntax, což snižuje náklady na udržování. Díky své jednoduchosti je Python vhodný pro nováčky ve světě programování, a tak je často vyučovaný jako první programovací jazyk. To ale neznamená, že je to pouze jazyk vhodný pro začátečníky. [13]

Python je velmi využívaný pro vývoj webových aplikací, vědeckých aplikací a machine learning. Je to především díky velkému množství knihoven, které jsou pro něj dostupné. Python je tedy jednoduchý, ale zároveň velmi schopný nástroj na řešení velkého množství problémů. [14]

2.4 Django

Django [15] je framework pro rychlou tvorbu bezpečných a udržitelných webových aplikací napsaný v Pythonu. Řeší velké množství problémů spojených s webovým vývojem, a tak se vývojáři mohou soustředit na důležitější části vývoje bez nutnosti vymýšlet něco, co již někdo dávno udělal.

Django se na rozdíl od Vue snaží být balíčkem nástrojů, který obsahuje všechno, co mohou vývojáři při práci potřebovat. To má velký vliv na bezpečnost výsledné

aplikace, protože mezi tyto nástroje patří například správa uživatelských účtů a hesel. Pokud tedy vývojář využije tuto funkcionalitu, je velká pravděpodobnost, že výsledná aplikace bude odolnější vůči útokům než v případě, kdy by se pokoušel o vlastní implementaci od základů. Aplikace je tak chráněná vůči velkému množství útoků jako jsou SQL Injection, *Cross-site scripting* (XSS) a další.[16]

Aplikace napsané v Djangu jsou také jednoduše škálovatelné, aby pokryly jakoukoliv poptávku. Je tomu tak především díky jasnému rozdělení různých částí aplikace, které je tak možné rozšiřovat podle potřeby. Tuto vlastnost v minuosti využily aplikace jako Instagram a Disqus, které zažily velký vzrůst počtu uživatelů. Tento framework je také multiplatformní, protože využívá Python, který běží na Windows, MacOS a Linuxu. [17]

2.5 Django REST framework

Tento framework je napsaný ve frameworku Django jako flexibilní sada nástrojů k vytváření REST API. Jedná se tedy o backendový framework sloužící jako prostředník mezi databází a klientem. Z Djanga používá funkce sloužící pro backend, protože API nepotřebuje funkcionalitu spojenou s tvorbou grafických uživateských rozhraní – *Graphical User Interface* (GUI).

REST

Representational State Transfer (REST) je sada doporučení pro tvorbu webových aplikačních rozhraní. RESTful rozhraní umožňuje přistupovat k datům na určitém místě pomocí 4 standardních metod HTTP. [18] [19]

Metoda GET slouží jako požadavek na stránku. Klient tak GET dotazem požádá server o stránku přítomnou na této URL a server mu ji vydá. Může se jednat o HTML stránku, webovou aplikaci, nebo jakýkoliv soubor či dokument.

Metoda POST slouží klientovi k vytvoření zdroje. Klient tedy odešle data serveru, který je následně vyhodnotí a případně uloží do databáze. Po uložení do databáze je možné k datům přistoupit metodou GET.

Metoda DELETE slouží k mazání zdrojů na příslušné URL adrese. Po tomto požadavku server zdroj z databáze vymaže a ten již nebude přístupný.

Metoda PUT slouží k aktualizaci již existujícího zdroje. Fungováním je podobná metodě POST.

Struktura Django REST API

Veškerá data jsou ukládána do tzv. modelů. Model je objekt, který představuje záznam v tabulce databáze a obsahuje veškerá data z tohoto záznamu. Definice

modelu v Django REST frameworku tedy vytvoří v databázi tabulku, do které se budou data jednotlivých objektů ukládat. Protože Django REST framework využívá funkce z Djanga, není nutné tabulky vytvářet ručně a ani psát SQL dotazy pro komunikaci s databází. Vše probíhá automaticky. Tabulky se všechny vytvoří tzv. migrací, která je nutná při změně struktury v modelech. [20]

Pro komunikaci přes REST API je třeba definovat zobrazení. Zobrazení naslouchá dotazům URL, které jsou přesně definovány, a následně vyhodnocuje, zda jsou dotazy správné a je tedy možné s nimi dále pracovat. [21]

Pro převádění dat mezi údaji v databázi a JSON dokumenty slouží serializéry. Serializéry mají definovaná data, která využívají z každého modelu, a která se také využívají ke kontrole dat, která na server odeslal klient. Pokud JSON dokument odeslaný klientem např. neobsahuje datovou položku, která je vyžadována, serializér vyhlásí chybu a požadavek je zamítnut. V případě, že je dokument správný, je možné v serializéru zavolat metodu .save(), která vygeneruje SQL dotaz, a tak tento nový objekt uloží do databáze. [22]

2.6 PostgreSQL

PostgreSQL je open-source objektově-relační databázový systém, který používá jazyk *Structured Query Language* (SQL). Vznikl z projektu Postgres na Kalifornské univerzitě v Berkeley a do podoby, jaká je používána dnes, se dostával přes 30 let. Za tyto roky si vybudoval reputaci díky své ověřené architektuře, spolehlivosti, škálovatelnosti, podpoře velkého množství datových typů a řadě funkcí. [23]

Splňuje požadavky *Atomicity, Consistency, Isolation, Durability* (ACID), které zajišťují platnost dat i přes chyby a další možná přerušení, jako jsou výpadky proudu. Data v databázi tedy zůstávají přesná i v případě podobných selhání.

Díky všem těmto vlastnostem je použitelný prakticky v jakémkoliv případě, ať už se jedná o malou aplikaci, jejíž databáze běží na jednom stroji, nebo rozsáhlé aplikace a systémy s obrovskými databázovými úložišti jako například Reddit, Instagram a Netflix. [24]

PostgreSQL je multiplatformní a běží na většině moderních operačních systémů jako například Linux, MacOS a Windows. Je napsaný v jazyce C, který se často používá v případech, kdy zařízení nemá příliš velký výkon, nebo je žádané, aby provádění akcí bylo co nejpřesnější a nejrychlejší.

2.7 Další použité nástroje

Kromě technologií přímo použitých pro implementaci je nutné si ještě představit nástroje používané ke zvýšení kvality této aplikace, ať už v kvalitě ve funkční stránce nebo v čitelnosti výsledného kódu. Jedná se především o nástroje sloužící k automatickému formátování kódu (Formattery) a nástroje kontroly používání best practices v kódu (Lintery).

Volby formatterů závisí hodně na preferencích, o kterých se vedou rozsáhlé debaty. V případě této práce byly zvoleny formattery jejichž výstup je dobře čitelný bez nutnosti velkých nastavení.

Prettier

Prettier je automatický formatter pro JavaScript, Vue, CSS, HTML, Markdown, Angular a další. V angličtině se podobné formattery označují jako *opinionated* (= tvrdohlavý), což vypovídá o jejich nastavitelnosti. Opinionated formaterry zpravidla nemají velké množství nastavitelných parametrů a díky tomu je výsledný zformátovaný kód konzistentní mezi různými prostředími, a tudíž daleko více čitelný. Starají se o to, aby týmy programátorů s různě zkušenými členy vyprodukovaly kód který bude formátovaný stejně bez ohledu na to, kdo jej napsal. [25]

Po zavolání Prettier proskenuje složku projektu a u všech podporovaných typů souborů zkontroluje, zda jsou zformátovány správně a případně formátování opraví.

Black

„By using Black, you agree to cede control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from pycodestyle nagging about formatting. You will save time and mental energy for more important matters.“ [26]

Black je nekompromisní automatický formatter pro Python. Jeho parametrů je ještě menší množství než u Prettier a díky tomu je ještě jednodušší začít ho používat. Code style, který opinionated formattery jako Prettier a Black vytvářejí, pomáhá soustředit se na řešení problémů místo zabývání se detaily jako jsou rozhodnutí, zda zalomit závorku na další řádek, nebo ji nechat tam, kde je. Problémy s formátováním tak mizí a výsledkem je konzistentní kód.

ESLint a Flake8

Tyto nástroje spadají do kategorie linterů. Lintery provádějí hlubší analýzu kódu a hledají problematické části, které nevyhovují code style nebo by mohly být problematické, či zbytečné. Díky tomu je možné odhalit velké množství problémů a opravit je ještě před spuštěním aplikace.

Kromě toho také lintery mohou upozorňovat na nepoužívané proměnné a zbytečně importované knihovny a funkce. Bývají daleko nastavitelnější než formattery, protože jejich hlavním cílem je na problémy upozorňovat a je tak možné je nastavit jako přísnější nebo benevolentnější podle potřeby. Lintery tedy provádí podobnou činnost jako formattery, ale na daleko hlubší úrovni a na problémy většinou pouze upozorňují. Formattery a lintery jsou tedy nástroje, které se vzájemně doplňují.[27]

ESLint [28] je linter programovacího jazyka JavaScript, zatímco *Flake8* [29] je zaměřený na Python.

Bandit

Bandit [30] také patří do kategorie linterů, ale na rozdíl od ESLint a Flake8 se Bandit soustředí pouze na běžné bezpečnostní chyby v Python kódu. Upozorňuje tedy například na používání nebezpečných funkcí jako je eval(), použití MD5 hashování u hesel a proměnné s názvy jako *SECRET_KEY*, které by měly být chráněny před zveřejněním.

Git

Git [31] je velmi rozšířený open-source verzovací systém používaný při vývoji aplikací. Každá verze (commit) obsahuje informace o změnách v souborech místo nutnosti kopírovat celý projekt.

pre-commit

Pre-commit [32] umožňuje spouštět skripty před každým git commitem. Je proto používán jako nástroj poslední kontroly projektu před vytvořením další verze a zajišťuje, že každý commit bude vždy správně zformátovaný a bez problematických částí. Pro potřeby tohoto projektu byl pre-commit nastaven tak, aby spouštěl všechny formattery a lintery. V případě objevení chyby se commit neprovede a je před další verzí nutné chybu odstranit.

3 Implementace webové aplikace

3.1 Návrh aplikace

Aplikace byla navržena tak, aby datový model umožňoval co největší modularitu, a proto byla data rozdělena do čtyř objektů a tím pádem i databázových tabulek. Datové modely tedy vyjadřují *Zranitelnost*, *Systém*, použitá *Protiopatření* a výsledné *Skóre*, které je vytvořeno kombinací tří předchozích modelů.

Tato modulárnost dat aplikace vede k možnosti různě modely kombinovat pro vytvoření responzivnějšího a provázanějšího systému. Každý systém tak může mít několik zranitelností, několik systémů může sdílet stejná protiopatření, ale bez nutnosti znovu ukládat data, která by byla společná. Navíc v případě, že se jeden tento společný prvek změní, změna se aplikuje na všechny závislé hodnoty skóre a skóre je pro ně přepočítáno.

Návrh tak představuje myšlenky zmíněné v části 1.2.6. V případě využití centrální databáze zranitelností by tak pouze stačilo přiřadit zranitelnosti jednotlivým systémům a nebylo by nutné jejich parametry zadávat. Taková hodnotící aplikace by značně zjednodušila hodnocení zranitelností přítomných na systémech v síti a zajistila by aktuálnost vlastností zranitelností v případě změny v centrální databázi.

Tento návrh je aplikovaný na backendu i frontendu aplikace. Na straně serveru se jedná primárně ukládání modelů do separátních databázových tabulek, ze kterých jsou vytvářeny objekty konkrétních prvků. Uživatelská strana aplikace zase slouží k předvedení této modularity uživateli, kde je možné jakkoliv prvky kombinovat podle potřeby.

3.2 Připravené vývojové prostředí

Veškerý vývoj probíhal v editoru Visual Studio Code (často zkracováno jako VS Code), který zdarma vydává firma Microsoft a funguje na operačních systémech Windows, MacOS a Linux. Jedná se o velmi *lightweight* nástroj, který ale obsahuje schopné nástroje, jež z něj dělají jeden z nejoblíbenějších vývojových editorů na světě.

Visual Studio Code podporuje velké množství programovacích jazyků, mezi než patří i JavaScript a Python, které byly využity při implementaci. Také obsahuje možnost přidání doplňků, které jeho funkcionalitu rozšiřují.

Do VS Code byly pro zkvalitnění práce přidány pluginy pro *Prettier*, *ESLint* a *Python*, které editor rozšiřují o podporu formatterů a linterů zmíněných v minulé kapitole. Dále byl nainstalován doplněk *Vetur*, který přidává do VS Code mnoho funkcí spojených s Vue.js jako například zvýraznění syntaxe, linting, našeptávání a další.

Vzdálený vývoj

Celý vývoj probíhal na vzdáleném virtuáním stroji s operačním systémem Ubuntu Server, ke kterému se připojovalo vzdáleně s pomocí pluginu *Remote - SSH*. Díky tomu byl celý vývoj oddělený od osobního počítače a probíhal tak přímo v prostředí na kterém byla aplikace později nasazena a které nemohly ovlivňovat žádné další aplikace. Samotná práce v editoru je potom kromě mírně delší doby načítání projektu prakticky nerozeznatelná od obyčejného vývoje přímo na fyzickém počítači.

Code style

Code style byl už částečně představený v minulé kapitole. O jeho konzistenci se starají formattery *Black* a *Prettier*. V případě backendu serveru, který je psaný v programovacím jazyce Python, byla nastavena délka řádku na 110 znaků pro dobrou čitelnost. *Prettier* byl nastaven na délku řádku 100 znaků, která byla naprosto dostačující a další argumenty byly nastaveny tak, aby se Vue.js část programu v Code style podobala Python části, aby čitelnost byla co nejvyšší.

Lintery

ESLint linter byl používán s několika pluginy. Kromě *eslint:recommended* byly použity *vue/vue3-essential*, *prettier/recommended* a *@vue/prettier*, převážně aby linter fungoval s funkcemi Vue.js a code style, který byl nastaven pro *Prettier*.

Ve Flake8 bylo nutné vypnout chybové kódy, které může způsobovat *Black*. Jedná se o kódy:

- *Continuation line under-indented for visual indent (E128)*
- *Whitespace before ':' (E203)*
- *Line break occurred before a binary operator (W503)*

Jsou spojené s doporučeními *pycodestyle* a *Black* code style je nedodržuje z důvodu lepší čitelnosti. Chybový kód *Line too long (E501)* byl vypnut z důvodu nastavení mírně větší délky řádku. Dále je nastaveno upozornění na přílišnou komplexitu funkcí, pokud funkce přesáhne cyklomatickou složitost 25.

3.3 Backend

Backend neboli serverová část aplikace je napsaný v programovacím jazyce Python a frameworku Django REST framework. Jak již bylo zmíněno, stará se i o zapisování a čtení dat z databáze. Bylo tedy nutné nainstalovat databázový systém PostgreSQL, založit v něm uživatele a databázi a údaje nutné k připojení vložit do parametrů v souboru *settings.py*.

Tento soubor obsahuje veškeré citlivé údaje jako heslo k databázi, tajný klíč, IP adresy, a proto byl pro demonstrační účely vytvořen výchozí soubor *default.settings.py*, ve kterém jsou tato citlivá data odstraněna, aby bylo možné alespoň nějakou verzi tohoto souboru přidat do gitu k verzování.

Django REST framework umožňuje vytvoření několika oddělených aplikací (každá se svými modely, serializéry a zobrazeními), pro potřeby této implementace byla ale vytvořena pouze jedna aplikace - *vulnerabilities*.

3.3.1 Modely

V souboru *models.py* byly vytvořeny čtyři základní modely: *Vulnerability*, *System*, *Countermeasures* a *Score*. Model *System*, který se skládá z velkého množství parametrů, byl pro větší přehlednost v kódu rozdělen na několik částí, tabulka v databázi se pro něj ale vytváří pouze jedna.

```
1  from django.db import models
2  from .choices import Vuln, Sys
3
4  class Vulnerability(models.Model):
5    name = models.CharField(max_length=140)
6    description = models.CharField(max_length=2048)
7
8    confidentiality_impact = models.FloatField(choices=Vuln.conf_impact)
9    integrity_impact = models.FloatField(choices=Vuln.int_impact)
10   availability_impact = models.FloatField(choices=Vuln.avail_impact)
11
12   skill_level = models.FloatField(choices=Vuln.skill_level)
13   required_privileges = models.FloatField(choices=Vuln.req_priv)
14   user_interaction = models.FloatField(choices=Vuln.user_interaction)
15
16   awareness = models.FloatField(default=0.5, choices=Vuln.awareness)
17   exploitability = models.FloatField(default=0.5, choices=Vuln.exploitability)
18   threat_intelligence = models.FloatField(default=0.5, choices=Vuln.threat_intel)
```

Výpis 3.1: Model Vulnerability

Na výpisu 3.1 je jako příklad uvedena definice modelu *Vulnerability*. Model tvoří proměnné typu Field a definují sloupce tabulky databáze a mimo jméno a popis se jedná o parametry z tabulek 1.4, 1.5 a 1.6. ID je automaticky vytvořený parametr, proto zde není uveden. Náleží každému prvku z jakékoli tabulky a je také automaticky inkrementován pro nové prvky, aby byl v rámci tabulky jedinečný. Ostatní modely jsou vytvořené obdobně. Výjimka je model *System*, který využívá abstraktní modely pro zlepšenou čitelnost. Model by jinak obsahoval přes 50 parametrů, ve kterých by bylo náročné se v programu orientovat.

V modelech byly použity proměnné typu *CharField* pro ukládání textu, konkrétně u jména a popisu, kde je možné nastavit hodnotu *max_length* pro omezení velikosti vstupu a snížení objemu databáze. Maximální délka jména je nastavena na 140 znaků. Zranitelnosti se zpravidla řídí pojmenováním *Common Vulnerabilities and Exposures* (CVE), u kterého by bylo zcela dostačujících i 20 znaků, u systémů a protiopatření už by tomu tak ale být nemuselo, a tak mají všechna jména nastavenou stejnou maximální délku z důvodu konzistence. Velikost popisu je nastavena maximálně na 2048 znaků, což by měl být více než dostatek prostoru pro uvedení dodatečných informací.

U všech parametrů s více možnými odpověďmi je použit *FloatField*, kam jsou ukládány přímo číselné hodnoty a pro odpovědi typu ano/ne je určen *BooleanField*. V případě, že je proměnná typu *FloatField*, je jí předán také seznam povolených hodnot, tak jak je to definováno v příslušných tabulkách.

U modelu skóre se používají tři proměnné typu *ForeignKey*, které odkazují na konkrétní prvek v jiné tabulce databáze. Jsou do nich ukládány odkazy na zranitelnost, systém a protiopatření, ze kterých se počítá výsledné skóre. Také obsahuje výsledné skóre, které je přepočítáno pouze v případě změny některého z odkázaných prvků.

3.3.2 Serializéry

Serializéry jsou třídy, které fungují jako prostředník mezi zobrazením a databází. V souboru *serializers.py* jsou vytvořené čtyři serializéry (podle počtu základních modelů). U těch lze manuálně vybrat, které parametry z databáze budou předávat. Zde se ale pracuje se všemi parametry modelu a je možné použít atribut *__all__* místo pracného jmenování každého z nich. Jmenování by bylo užitečné v případě, kdyby bylo nutné posílat pouze některé parametry. Lze také přidat parametr, který se v modelu nevyskytuje.

Na výpisu 3.2 je představen serializér modelu zranitelnosti. Všechny ostatní serializéry jsou vytvořeny obdobným způsobem.

```python
from rest_framework import serializers
from .models import Vulnerability, Countermeasures, Score, System

class VulnerabilitySerializer(serializers.ModelSerializer):
    class Meta:
        model = Vulnerability # z jakeho modelu brat atributy
        fields = "__all__" # vsechny atributy
```

Výpis 3.2: Příklad serializérů

3.3.3 Výpočet skóre

Výpočet skóre je implementovaný v souboru *score.py*. Probíhá ve 4 částech, přesně tak, jak je to popsáno v podsekci 1.2.5. Je volán v souboru *views.py* při uložení nového skóre a při aktualizování jakéhokoliv jiného prvku jsou aktualizována všechna skóre, která obsahují změněný prvek. Funkce *calculate_score* vyžaduje parametry *Vulnerability*, *System* a *Countermeasures*, aby hodnota mohla být vypočítána.

3.3.4 Zobrazení a URL

Ve výpisu 3.3 je představen soubor *urls.py* s definovanými URL adresami, na kterých API naslouchá, a funkce, která je zavolána v případě dotazu na takovou adresu. Tyto funkce se nachází v souboru *views.py*.

```python
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r"^api/vulnerability-list/?$", views.vulnerability_list),
    url(r"^api/vulnerability/(?P<pk>[0-9]+)/?$", views.vulnerability_detail),
    url(r"^api/system-list/?$", views.system_list),
    url(r"^api/system/(?P<pk>[0-9]+)/?$", views.system_detail),
    url(r"^api/countermeasures-list/?$", views.countermeasures_list),
    url(r"^api/countermeasures/(?P<pk>[0-9]+)/?$", views.countermeasures_detail),
    url(r"^api/score-list/?$", views.score_list),
    url(r"^api/score/(?P<pk>[0-9]+)/?$", views.score_detail),
]
```

Výpis 3.3: Soubor urls.py

Funkce používají dekorátor `@api_view([])`, ve kterém je specifikováno, kterým HTTP metodám funkce naslouchá. Jako příklad je ve výpisu 3.4 představena funkce *system_detail*, která naslouchá dotazům na konkrétní záznam z tabulky *System*. Práce s metodou *POST*, která je používaná v jiném zobrazení funguje obdobně jako práce s metodou *PUT* zde. Pouze se SystemSerializeru nepředává objekt záznamu (zde *system*), v tom případě je při zavolání metody *save()* vytvořen nový záznam.

```
@api_view(["GET", "PUT", "DELETE"])
def system_detail(request, pk):
    # najde systém podle pk (=id prvku), ktere je definovane tvarem URL v~urls.py
    try:
        system = System.objects.get(pk=pk)
    except System.DoesNotExist:
        return JsonResponse(
            {"message": "The system does not exist"},
            status=status.HTTP_404_NOT_FOUND,
        )

    # GET / PUT / DELETE system
    if request.method == "GET":
        system_serializer = SystemSerializer(system)
        return JsonResponse(system_serializer.data)
    elif request.method == "PUT":
        system_data = JSONParser().parse(request)
        system_serializer = SystemSerializer(system, data=system_data)
        if system_serializer.is_valid():
            obj = system_serializer.save() # uloží změněná data systému
            # najde všechny skóre se stejným systémem a přepočítá jejich skóre
            scores = Score.objects.filter(system__id=obj.id)
            for score in scores:
                score.score = ScoreCalculator.calculate_score(
                    score.vulnerability, score.system, score.countermeasures
                )
                score.save()
            return JsonResponse(system_serializer.data)
        return JsonResponse(system_serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    elif request.method == "DELETE":
        system.delete()
        return JsonResponse(
            {"message": "System was deleted successfully!"},
            status=status.HTTP_204_NO_CONTENT,
        )
```

Výpis 3.4: Zobrazení system_detail

3.4 Frontend

Frontend neboli klientská část aplikace je napsaný ve frameworku Vue.js. Základem frontendu je soubor *src/main.js*, do kterého se vkládají všechny doplňky a aplikace zde začíná. Dalším důležitým souborem je *App.vue*, který představuje hlavní Vue komponent. Do něj jsou vloženy komponenty *Header* s logem a názvem aplikace a *Menu*, které obsahuje router odkazy na důležité podstránky. Druhým je komponent *router-view*, o který se stará Vue Router a v jeho obsahu přepíná mezi různými komponenty podle nastavení v souboru *src/router/index.js*.

Soubor *variables.js* obsahuje jedinou proměnnou, kterou je *apiLink*, což je základ adresy API. Ten přebírají všechna místa, kde probíhá volání na API. Řeší tak problémy, které by vznikaly, pokud by na různých strojích byly adresy pro API různé. Je to jediná proměnná, kterou je třeba změnit a veškerá volání API se změní na jinou adresu. Společně se souborem *vue.config.js* byly vynechány, protože obsahují data, která mohou být citlivá pro git repozitář a stejně jako *settings.py* v backendu byly nahrazeny soubory s předponou *default* a základní konfigurací.

Router obsahuje cesty k přepínaným komponentům a přepíná je podle specifikované URL adresy. Na výpisu 3.5 je příklad, kdy router obsahuje cesty ke *ScoreList*, pro výpis celého seznamu objektů *Skóre*, a *ScoreItem* pro zobrazení detailů konkrétního skóre podle jeho ID. Také je zde vidět dynamická změna titulku stránky podle názvu zobrazené komponenty.

```
Vue.use(Router)
const routes = [ // pole cest routeru
  {
    path: "/score-list", // URL cesty
    name: "Score List", // název podstránky
    component: ScoreList, // zobrazený komponent
  },
  {
    path: "/score/:id", // URL cesty s parametrem id
    name: "Score Item",
    component: ScoreItem,
    props: (route) => { // předání parametru id komponentu, pokud je číselný
      const id = Number.parseInt(route.params.id, 10)
      if (Number.isNaN(id)) {
        return 0
      }
      return { id }
  },
]
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
})
const DEFAULT_TITLE = "Vulnerability Calculator"
router.beforeEach((to, _from, next) => {
  document.title = to.name + " - " + DEFAULT_TITLE
  next()
})
```

Výpis 3.5: Vue Router

3.4.1 Vuex Store

Store je centrální místo, kam se ukládají data, se kterými aplikace pracuje. Díky tomu není nutné data předávat mezi jednotlivými komponentami, což často vede ke komplikovanému a nečitelnému kódu aplikace, se kterým se špatně pracuje. V této práci je *Store* rozděleno do pěti souborů – jeden hlavní a čtyři pro jednotlivé typy objektů. Všechny tyto soubory se nachází ve složce *src/store*.

Store obsahuje pět částí:

- `state` – definuje data které *Store* obsahuje
- `mutations` – obsahuje funkce které mění data uložená ve `state`
- `actions` – obsahuje asynchronní funkce pro práci s API
- `getters` – obsahuje funkce které vrací data ve `state`
- `modules` – slouží k importování dalších souborů *Store*. Zde je využita pouze v hlavním souboru.

3.4.2 Předlohy objektů

Protože se v aplikaci pracuje s daty na různých místech, byl ve složce *src/data* vytvořen soubor pro každý z objektů v aplikaci. Obsahuje předlohu proměnných, které každý objekt obsahuje. Soubory *system.js* a *vulnerability.js* ještě obsahují objekt *Options*, ve kterém jsou vypsány všechny možnosti, které jejich proměnné mohou nabývat, pokud se nejedná o text nebo boolean.

Každá taková proměnná obsahuje pole možných hodnot, kterých může nabývat a také popisek, který značí, co daná hodnota znamená, viz výpis. Options jsou využity ve formulářích, kde jsou z nich generovány možnosti pro komponenty `select`.

Výpis 3.6 ukazuje příklad pro předlohu objektu Vulnerability s možnostmi jedné jeho proměnné.

```
export default class Vulnerability {
  id = null
  name = ""
  description = ""
  confidentiality_impact = null
  integrity_impact = null
  availability_impact = null
  skill_level = null
  required_privileges = null
  user_interaction = null
  awareness = null
  exploitability = null
  threat_intelligence = null
}

export const Options = {
  confidentiality_impact: [
    { label: "All data can be stolen", value: 1.0 },
    { label: "Large amount of critical data can be stolen", value: 0.78 },
    { label: "Large amount of noncritical data can be stolen", value: 0.67 },
    { label: "Small amount of critical data can be stolen", value: 0.67 },
    { label: "Small amount of noncritical data can be stolen", value: 0.22 },
  ],
}
```

Výpis 3.6: Předloha objektu Vulnerability s příkladem možností jedné proměnné

3.4.3 Struktura komponentů

Komponenty jsou v projektu rozdělené do složek. Složka *views* obsahuje komponenty přepínané doplňkem Vue Router rozdělené dále na složky *forms* (formuláře), *items* (detaily konkrétních objektů) a *lists* obsahující seznamy objektů. Ve složce *components* se nachází ostatní stavební bloky, které využívají komponenty *views*. Konkrétně se jedná o tabulky se seznamy objektů, *Header* a *Menu*.

Jednotlivé komponenty jsou si velice podobné, pro představení jejich struktury je využit komponent *Home*, který představuje domovskou stránku aplikace. Tento komponent je pomocí routeru vsazen do komponentu *App*, který už obsahuje Header a Menu a není tak nutné je přidávat do routerem vsazovaných komponentů.

Na výpise 3.7 lze jasně sledovat HTML strukturu komponentu v elementu `template`, element `script`, který obsahuje v tomto případě pouze název komponentu a element `style` s CSS pro nastavení vzhledu komponentu. Příznak *scoped* značí, že se CSS vztahují pouze na tento komponent.

V případě, že sem byly vloženy další komponenty, které by měly dědit některé CSS záznamy, přidal by se nový element `style`, tentokrát bez *scoped* příznaku. Případné vkládané komponenty by se importovaly v části `script` a vkládaly do `template`. Do *export default{}* se také mohou vkládat JavaScriptové funkce pro načítání dat a práci s nimi.

```
<template>
  <div class="route-contents">
    <img alt="App logo" src="../../public/logo_full_200px.png" />
    <h1>Welcome to the VulnCalc app</h1>
    <p>
      This app is an implementation of the David Pecl Scoring System (DPSS).
      It can be used to assess the vulnerabilities present on a network
      and determine the order in which vulnerabilities should be handled.
    </p>
    <!-- další odstavce jsou vynechány z ukázky -->
  </div>
</template>

<script>
export default {
  name: "Home",
}
</script>

<style scoped>
img {
  max-width: 100%;
}
.route-contents p {
  overflow-wrap: break-word;
}
</style>
```

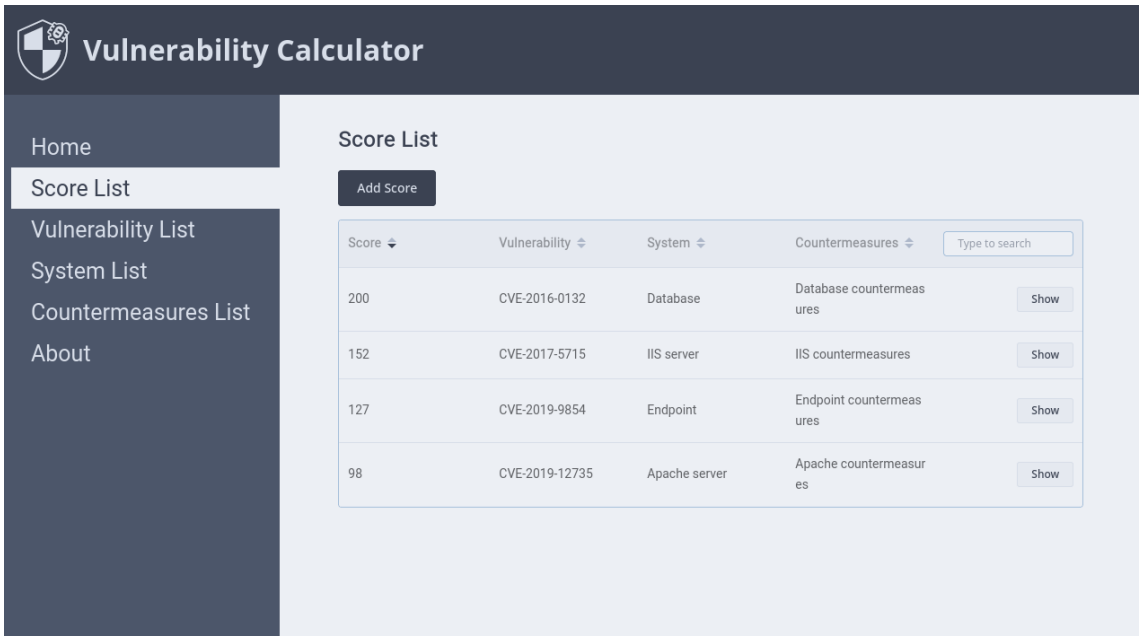
Výpis 3.7: Struktura komponentu Vue.js

3.4.4 Grafické rozhraní

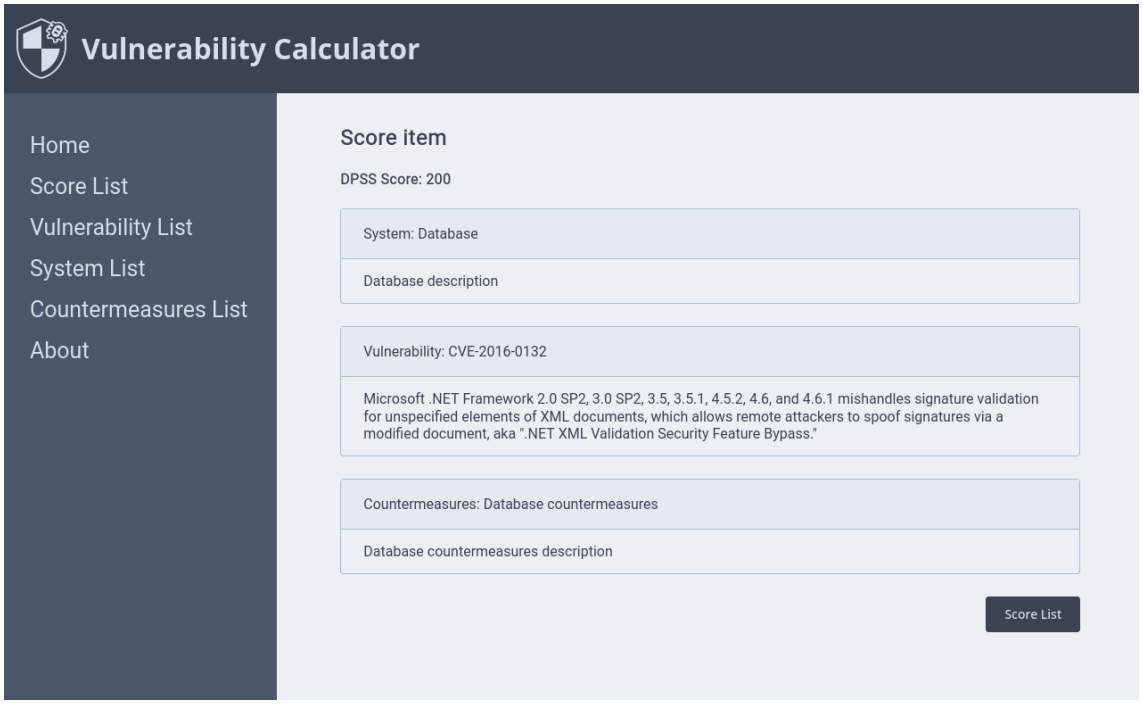
Rozložení grafického rozhraní bylo částečně představeno společně s komponenty. V této části bude představena mapa aplikace a grafický styl. Na obrázku 3.1 je vidět hlavní stránka aplikace. V horní části obrázku je vidět *Header* s logem aplikace, na levé straně se nachází *Menu* s odkazy na nejdůležitější části aplikace a ve zbylém prostoru je samotný router pohled na komponent *Home*.

Obr. 3.1: GUI – Hlavní stránka

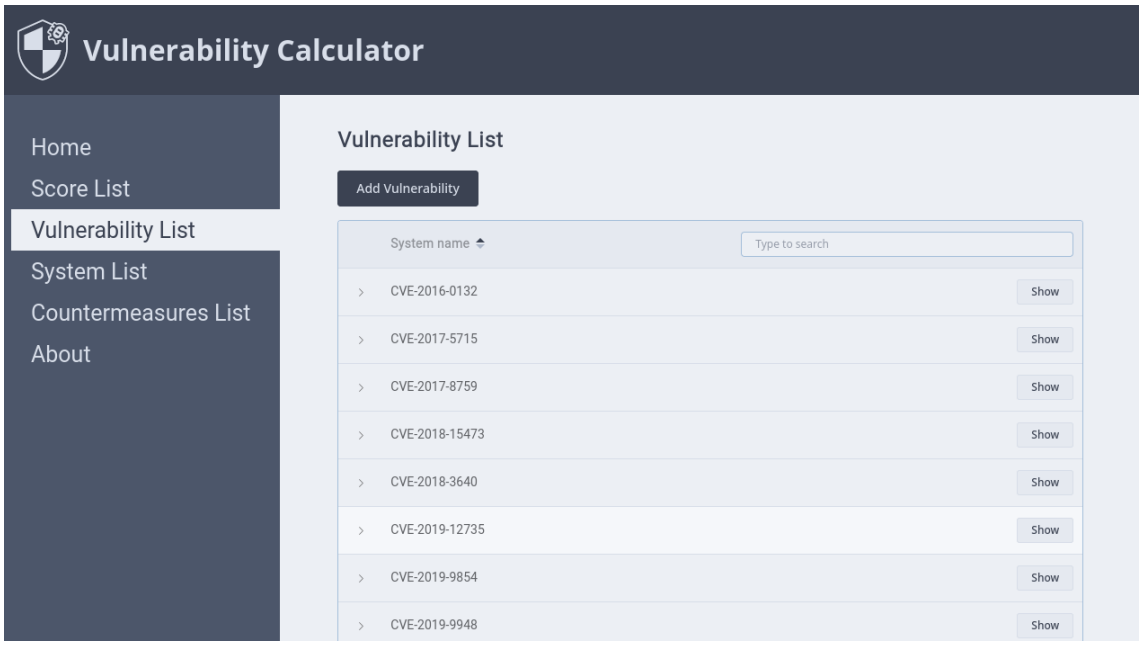
Stránka *Score List* obsahuje seznam hodnot skóre v tabulce, kde lze řadit hodnoty podle skóre a vyhledávat. S pomocí tlačítka *Add Score* lze přidávat nové hodnoty, tlačítko *Show* u každého prvku zobrazí detailnější stránku s více informacemi o konkrétním skóre. Tyto stránky lze vidět na obrázcích 3.2 a 3.3. Obrázek 3.4 zobrazuje ještě seznam zranitelností, který je velmi podobný jako u skóre, v tabulce lze ale navíc zobrazit popis zranitelnosti kliknutím na šipku na levé straně tabulky. Ostatní seznamy jsou navrženy stejně jako je tomu u seznamu zranitelností.



Obr. 3.2: GUI – Seznam hodnot skóre

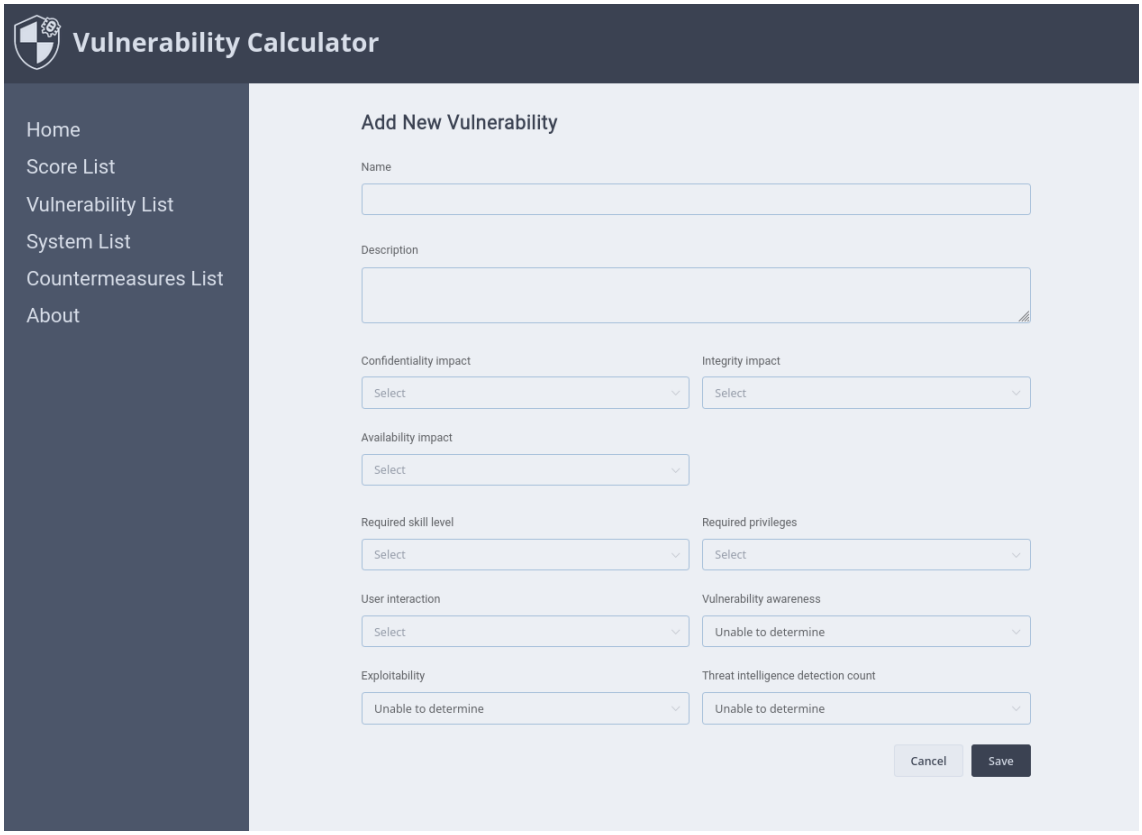


Obr. 3.3: GUI – Detail Skóre

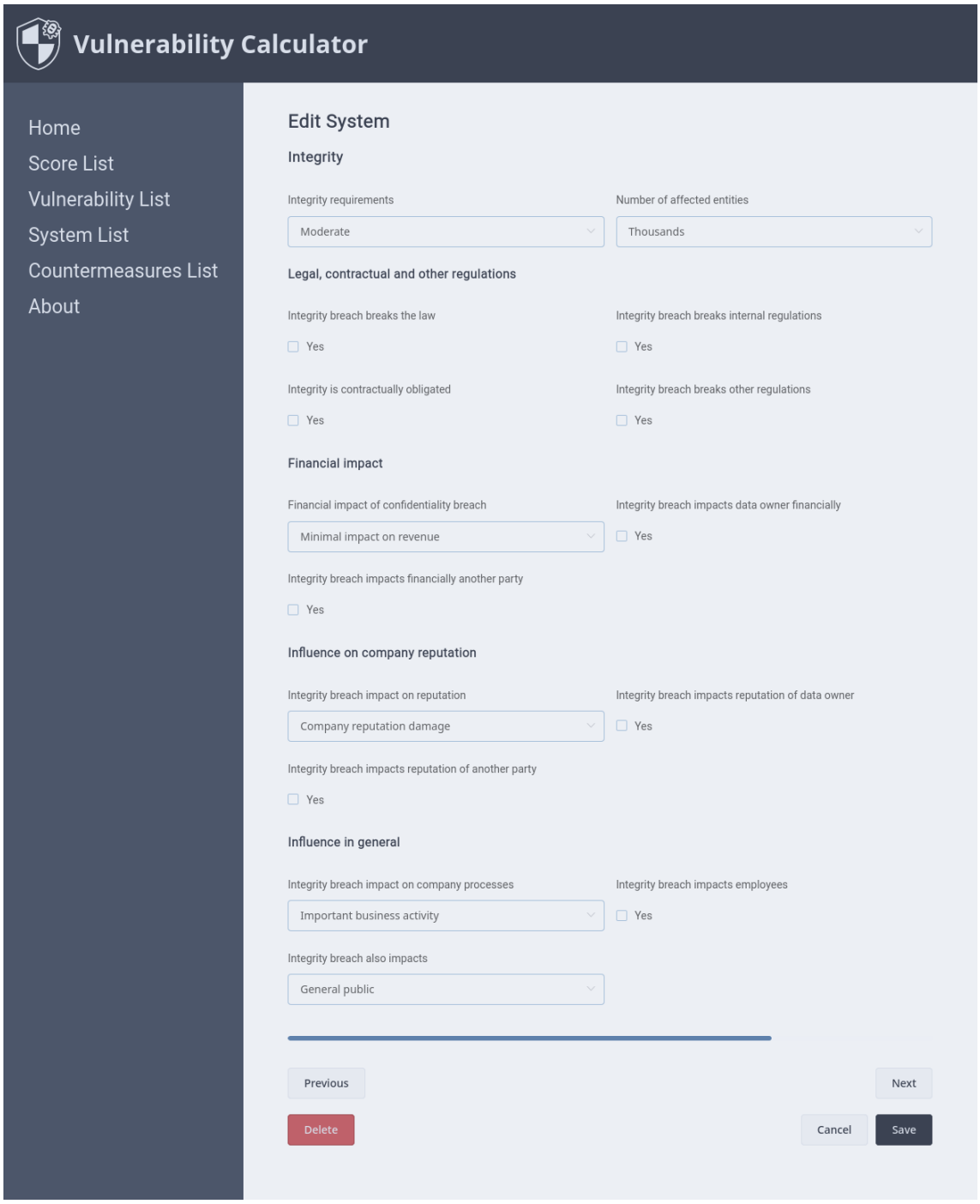


Obr. 3.4: GUI – Seznam zranitelností

Po kliknutí na tlačítko *přidat zranitelnost* je zobrazen formulář, kam je nutné vyplnit detaily přidaného prvku a data jsou odeslána na backend k uložení do databáze. Formulář je představen na obrázku 3.5 a veškeré ostatní formuláře jsou obdobné, s výjimkou systému, který má příliš mnoho parametrů a bylo nutné formulář pro lepší přehlednost rozdělit na části. Každý prvek lze také upravit nebo smazat. Smazat lze prvek ze stránky editace, na kterou vede tlačítko na stránce detailu prvku. Obrázek 3.6 zobrazuje editační stránku systému s viditelným rozdělením formuláře – ukazatel průběhu ve spodní části společně s tlačítky *zpět* a *další*.



Obr. 3.5: GUI – Formulář přidání nové zranitelnosti



Obr. 3.6: GUI – Formulář editace systému

Závěr

Cílem bakalářské práce bylo analyzovat současný stav problematiky hodnocení zranitelností, realizovat návrh kalkulátoru a provést implementaci celé aplikace. V rámci analýzy stavu problematiky byly popsány metodologie CVSS a nově navržená metoda Davida Pecla.

Druhá kapitola popisuje technologie použité při implementaci – JavaScript, Vue.js, Python, Django REST Framework a veškeré ostatní knihovny a nástroje které byly při implementaci použity.

Třetí kapitola se věnuje implementaci aplikace – odůvodnění datového návrhu aplikace, popis programové struktury a dále vzhledu a mapě grafického uživatelského rozhraní.

Implementace aplikace navazuje na předešlou semestrální práci, kterou značně rozšiřuje co do funkcionality a uživatelské přívětivosti. Backend aplikace obsahuje kontrolu vstupu proti neplatným datům, frontend využívá centrální správu dat. Grafické rozhraní bylo výrazně vylepšeno oproti předchozí verzi.

Aplikace byla psána s pomocí linterů a formatterů, dodržuje zásady obou využitých jazyků, objektově orientovaný přístup a má jednotný code style.

Literatura

[1] PECL, D. *Návrh metody pro hodnocení bezpečnostních zranitelností systémů.* [online]. Brno, 2020. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/125988>. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. [cit. 5. 12. 2020].

[2] FIRST *Common Vulnerability Scoring System SIG* [online]. 2021 [cit. 26. 5. 2021]. Dostupné z URL: <https://www.first.org/cvss>

[3] FIRST *Common Vulnerability Scoring System version 3.1: Specification Document: CVSS Version 3.1 Release* [online]. 2020 [cit. 29. 11. 2020]. Dostupné z URL: <https://www.first.org/cvss/v3.1/specification-document>

[4] MDN WEB DOCS *About JavaScript* [online]. 4. března 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>

[5] OPENJS FOUNDATION *Electron Apps* [online]. 2021 [cit. 30. 5. 2021]. Dostupné z URL: <https://www.electronjs.org/apps>

[6] CROMWELL, V. *Between the Wires: An interview with Vue.js creator Evan You* [online]. 30 května 2017 [cit. 6. 12. 2020].
Dostupné z URL: <https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/>

[7] YOU, E. *Introduction* [online]. 2020 [cit. 30. 5. 2021]. Dostupné z URL: <https://v3.vuejs.org/guide/introduction.html>

[8] YOU, E.; MOROTE, E. *Vue router* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://router.vuejs.org/>

[9] SCHEPENAAR, W. *Server-side vs Client-side Routing* [online]. 29. května 2017 [cit. 6. 12. 2020]. Dostupné z URL: <https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f>

[10] VUEX *What is Vuex?* [online]. 2021 [cit. 26. 5. 2021]. Dostupné z URL: <https://vuex.vuejs.org/>

[11] SARJEANT, J. *Promise based HTTP client for the browser and node.js* [online]. 3. prosince 2020 [cit. 30. 5. 2021]. Dostupné z URL: <https://axios-http.com/>

[12] ELEMENT TEAM *A Desktop UI Library* [online]. 2021 [cit. 26. 5. 2021]. Dostupné z URL: <https://element-plus.org/#/en-US>

[13] PYTHON SOFTWARE FOUNDATION *What is Python? Executive Summary* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://www.python.org/doc/essays/blurb/>

[14] WARCHOL, K. *What Is Python Used For? 5 Industries That Can't Do Without It* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://skillcrush.com/blog/what-is-python-used-for/>

[15] DJANGO SOFTWARE FOUNDATION *Meet Django* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://www.djangoproject.com>

[16] DJANGO SOFTWARE FOUNDATION *Security in Django* [online]. 2020 [cit. 30. 5. 2021]. Dostupné z URL: <https://docs.djangoproject.com/en/3.2/topics/security/>

[17] MDN WEB DOCS *Django introduction* [online]. 26. října 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>

[18] RESTFULAPI.NET *What is REST* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://restfulapi.net/>

[19] MDN WEB DOCS *HTTP request methods* [online]. 2. prosince 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

[20] DJANGO SOFTWARE FOUNDATION *Models* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://docs.djangoproject.com/en/3.1/topics/db/models/>

[21] DJANGO REST FRAMEWORK *Class-based Views* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://www.django-rest-framework.org/api-guide/views/>

[22] DJANGO REST FRAMEWORK *Serializers* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://www.django-rest-framework.org/api-guide/serializers/>

[23] POSTGRESQL GLOBAL DEVELOPMENT GROUP *What is PostgreSQL* [online]. 2021 [cit. 30. 5. 2021]. Dostupné z URL: <https://www.postgresql.org/about/>

[24] STACKSHARE *PostgreSQL* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://stackshare.io/postgresql/>

[25] LONG, J. et al. *Why Prettier?* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://prettier.io/docs/en/why-prettier.html>

[26] LANGA, Ł. et al. *The uncompromising code formatter* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://black.readthedocs.io/en/stable/>

[27] GUIMARÃES, G. *What is a linter and why your team should use it?* [online]. 3. července 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://sourcelevel.io/blog/what-is-a-linter-and-why-your-team-should-use-it>

[28] OPENJS FOUNDATION et al. *Find and fix problems in your JavaScript code* [online]. 2020, July 3 [cit. 6. 12. 2020]. Dostupné z URL: <https://eslint.org/>

[29] CORDASCO, I. *Flake8: Your Tool For Style Guide Enforcement* [online]. 2016 [cit. 6. 12. 2020]. Dostupné z URL: <https://flake8.pycqa.org/en/latest/>

[30] BANDIT DEVELOPERS *Bandit* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://github.com/PyCQA/bandit>

[31] CHACON, S. et al. *Getting Started - What is Git?* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>

[32] FINKLE, M. et al. *Introduction* [online]. 2020 [cit. 6. 12. 2020]. Dostupné z URL: <https://pre-commit.com/>

Seznam symbolů a zkratek

API	Application Programming Interface
ACID	Atomicity, Consistency, Isolation, Durability
CIA	Confidentiality, Integrity, Availability
CSS	Cascading Style Sheets
CVSS	Common Vulnerability Scoring System
CVSSv3	Common Vulnerability Scoring System verze 3
CVE	Common Vulnerabilities and Exposures
DB	Databáze
DOM	Document Object Model
FIRST	Forum of Incident Response and Security Teams
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
DPSS	David Pecl Scoring System
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locatoe
VS	Visual Studio
XSS	Cross-site scripting

Seznam příloh

A Návod ke spuštění aplikace

Následující návod platí pouze pro spuštění na lokálním stroji ve vývojářském režimu. V případě nasazení aplikace v produkčním režimu je nutné sestavit produkční verzi frontendu a dále nainstalovat a nakonfigurovat HTTP server (Apache, nginx, ...).

A.1 Příprava

Pro správné fungování aplikace je nutné mít nainstalované základní balíčky:

```
apt -y install python3 python3-pip npm postgresql
npm -g install vue@next
python3 -m pip install -r requirements.txt
export PATH="$HOME/.local/bin:$PATH"
```

Výpis A.1: Instalace základních balíků

Z předloh je třeba vytvořit konfigurační soubory:

```
cp backend/VulnCalc/default.settings.py backend/VulnCalc/settings.py
cp frontend/default.variables.js frontend/variables.js
cp frontend/default.vue.config.js frontend/vue.config.js
```

Výpis A.2: Vytvoření konfiguračních souborů

V *backend/VulnCalc/settings.py* je nutné do položky *SECRET_KEY* uložit tajný klíč, lze využít například on-line generátor `https://djecrety.ir/`.

Dále je potřeba vytvořit PostgreSQL databázi a její údaje zapsat do souboru *backend/VulnCalc/settings.py* do pole *DATABASES* (výchozí nastavení – uživatel: *vulncalc*, heslo: *vulncalc* a název databáze: *vulndb*).

```
sudo su - postgres
createuser --pwprompt vulncalc
psql -c "CREATE DATABASE vulndb;"
exit
```

Výpis A.3: Konfigurace databáze

Následuje migrace modelů do databáze:

```
python3 backend/manage.py migrate main
python3 backend/manage.py migrate
```

Výpis A.4: Migrace modelů do databáze

Nainstalují se potřebné *npm* balíky:

```
cd frontend/
npm install
cd ..
```

Výpis A.5: Instalace dodatečných npm balíků

V souboru *frontend/variables.js* je nutné upravit adresu tak, aby odpovídala adrese backendu.

Tím je aplikace připravena ke spuštění.

A.2 Spuštění aplikace

Backend aplikace se na lokálním zařízení spouští příkazem:

```
python3 backend/manage.py runserver 8080
```

Výpis A.6: Spuštění backendu

Následuje spuštění frontendu:

```
cd frontend/
npm run serve
```

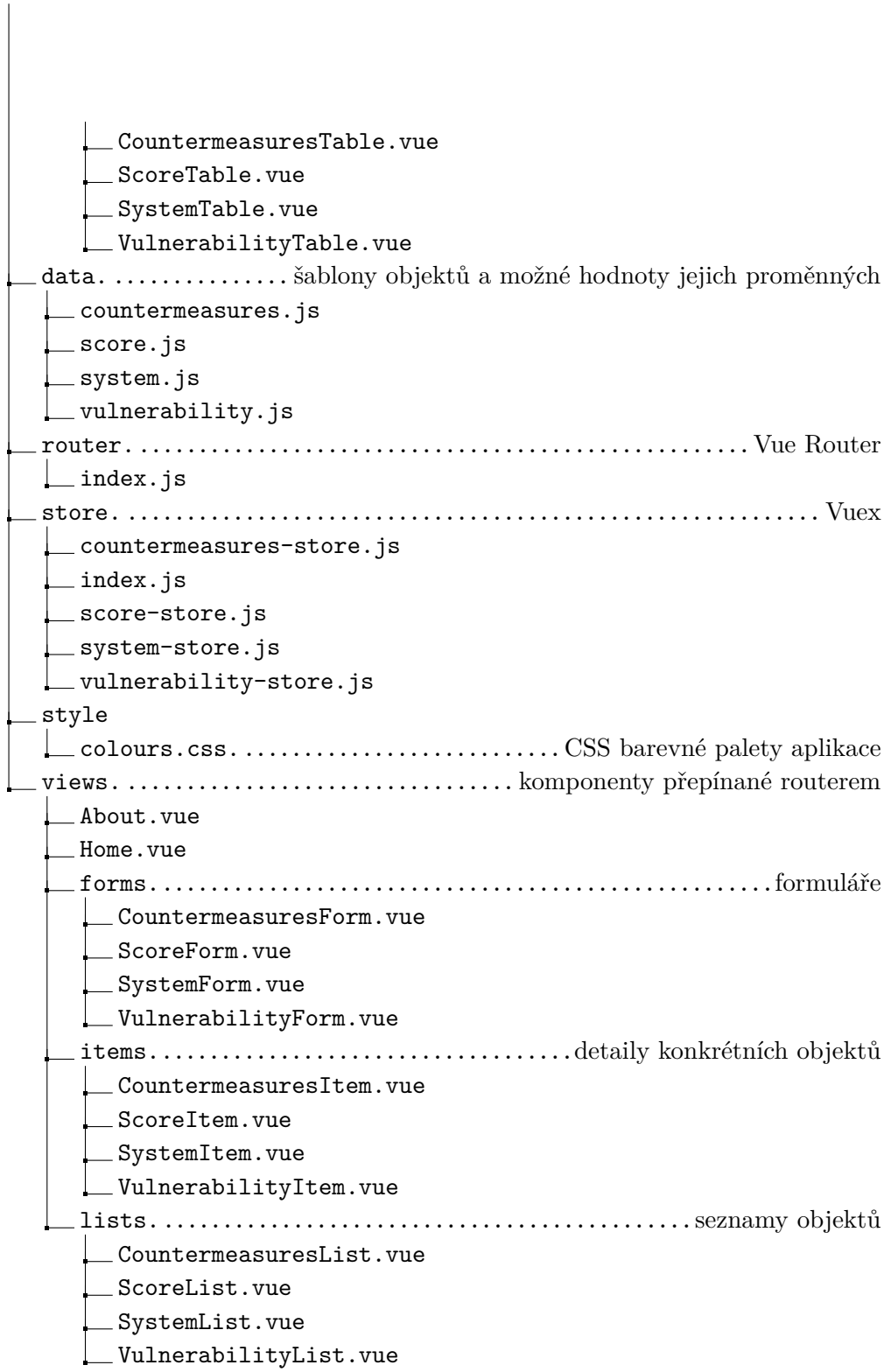
Výpis A.7: Spuštění frontendu

Terminál po chvíli vypíše adresu a port, ty stačí zadat do prohlížeče do pole URL a potvrdit.

B Kód aplikace

Přiložený archiv obsahuje kód aplikace *VulnCalc* v době odevzdání aplikace. V kořenové složce je obsažen soubor README.md s instrukcemi pro spuštění aplikace a soubory s nastaveními vývojového prostředí. Ve výpisu struktury složky s aplikací byly vynechány vygenerované soubory, které v rámci implementace nebyly upravovány. Tyto soubory jsou ale obsaženy v archivu.

- `.flake8` — nastavení Flake8
- `.pre-commit-config.yaml` — nastavení pre-commit
- `pyproject.toml` — nastavení Black
- `README.md` — instrukce pro spuštění aplikace
- `requirements-dev.txt` — python balíčky pro vývoj
- `requirements.txt` — python balíčky pro běh aplikace
- `vetur.config.js` — nastavení rozšíření Vetur
- `backend` — Django REST framework část popsaná v části 3.3
  - `manage.py` — hlavní ovládací soubor
  - `main` — složka s hlavními soubory backendu
    - `choices.py` — povolené hodnoty polí
    - `models.py` — databázové modely
    - `score.py` — výpočet skóre
    - `serializers.py` — obsahuje serializéry
    - `urls.py` — nastavení naslouchání URL
    - `views.py` — API zobrazení
    - `migrations` — automaticky generované migrační soubory pro databázi
  - `VulnCalc`
    - `default.settings.py` — základní nastavení backend serveru
- `frontend` — Vue.js část
  - `.eslintrc.js` — nastavení ESLint
  - `.prettierrc.js` — nastavení Prettier
  - `default.variables.js` — proměnná s API odkazem
  - `default.vue.config.js` — základní nastavení Vue.js
  - `package.json` — použité npm balíčky
  - `public` — složka s veřejnými soubory jako jsou obrázky
    - `favicon.png`
    - `index.html`
    - `logo_70px.png`
    - `logo_full_200px.png`
  - `src` — hlavní složka frontendu
    - `App.vue` — hlavní Vue komponent
    - `main.js` — hlavní soubor frontendu
    - `components` — komponenty používané ve views
      - `Header.vue` — komponent Header
      - `Menu.vue` — komponenty Menu
      - `tables` — komponenty tabulek


CountermeasuresTable.vue
ScoreTable.vue
SystemTable.vue
VulnerabilityTable.vue
data. . . . . šablony objektů a možné hodnoty jejich proměnných
countermeasures.js
score.js
system.js
vulnerability.js
router. . . . . Vue Router
index.js
store. . . . . Vuex
countermeasures-store.js
index.js
score-store.js
system-store.js
vulnerability-store.js
style
colours.css. . . . . CSS barevné palety aplikace
views. . . . . komponenty přepínané routerem
About.vue
Home.vue
forms. . . . . formuláře
CountermeasuresForm.vue
ScoreForm.vue
SystemForm.vue
VulnerabilityForm.vue
items. . . . . detaily konkrétních objektů
CountermeasuresItem.vue
ScoreItem.vue
SystemItem.vue
VulnerabilityItem.vue
lists. . . . . seznamy objektů
CountermeasuresList.vue
ScoreList.vue
SystemList.vue
VulnerabilityList.vue